Multi-Core Processor Memory Contention Benchmark Analysis Case Study

Tyler Simon, Computer Sciences Corp.
James McGalliard, FEDSIM

Abstract:

Multi-core processors dominate current mainframe, server, and high performance computing (HPC) systems. This paper provides synthetic kernel and natural benchmark results from an HPC system at the NASA Goddard Space Flight Center that illustrate the performance impacts of multi-core (dual- and quad-core) vs. single core processor systems. Analysis of processor design, application source code, and synthetic and natural test results all indicate that multi-core processors can suffer from significant memory subsystem contention compared to similar single-core processors.

1. INTRODUCTION

Moore's Law predicts that the performance of central processing units (CPUs) doubles approximately every 18 months. This prediction has held true for about 40 years[1]. Among the many advances in CPU design and construction, the single most important factor fulfilling Moore's prediction is manufacturers' ability to fabricate chips with narrower circuit paths, pack transistors more densely and allow the processor clock to speed up. Until recently.

Recent generation microprocessors suffer increasingly from transistor current leakage due to the narrow width of circuit paths and the insulation between them. As leakage increases, processors draw more power and generate more heat, to the point that significant further reductions in chip size and increases in clock speed may not be feasible with current technology.

Manufacturers have turned to multiple-core processors in response. With two, four or more CPU cores packed on a single chip, the chip's theoretical peak performance continues to follow the common interpretation of Moore's prediction. However, for many real workloads, when essentially the same memory subsystem has to support twice or four times as many instructions per second as previously, memory becomes a significant bottleneck and peak performance is not achievable.

This paper, based on synthetic kernel and natural benchmark tests run primarily on a highly parallel Linux cluster supercomputer located in the NASA Center for Computational Sciences[2] (NCCS) at the Goddard Space Flight Center in Greenbelt, Maryland, illustrates the phenomenon of multi-core processor memory contention.

The NCCS User Services Group (USG) administers accounts and allocations, runs the Help Desk and trouble ticketing and monitors system status. Software engineering and technical assistance is generally provided by the Software Integration and Visualization Office (SIVO). These benchmark results from USG and SIVO technical staff are intended to help users understand the behavior of multi-core systems and optimize their codes.

The paper is structured as follows:

- Section 1 describes the NCCS environment, major workloads and HPC systems.
- Section 2 discusses recent generation multi-core processor design.
- Section 3 provides synthetic kernel and natural application benchmark test results.
- Section 4 discusses some results from other studies.
- Section 5 provides conclusions and comments.

1.1 NCCS ENVIRONMENT

Goddard is a major center for NASA's Science Mission Directorate and is home to the nation's largest community of Earth scientists and engineers. Goddard's missions include expansion of knowledge of the Earth and its environment, the solar system, and the universe through observations from space. The Hubble Space Telescope was designed and built

[1] This is the common interpretation of Moore's Law, which actually states that the number of transistors on a given integrated circuit will double approximately every two years [Moore].

[2] *The authors would like to acknowledge the support of the NASA Center for Computational Sciences in the development of this paper, including the use of NCCS systems for benchmark tests.*

at Goddard, and it is a design center for Earth-observing satellites and other spacecraft. Goddard is also the home of the NCCS.

NCCS is a supercomputing data center that provides Goddard's science community with HPCs, mass storage, network infrastructure, software and support services. About 600 scientists use NCCS systems to increase their understanding of the Earth and space through computational modeling and processing of space-borne observations. NCCS systems are targeted to the specialized needs of Earth and space scientists and NASA's exploration initiative. NCCS performance management was the subject of a 2003 CMG paper [Glassbrook].

1.2 NCCS WORKLOADS

The largest NCCS workloads are mathematical models of the Earth's atmosphere, oceans and climate. One important constituent of this workload is data assimilation, which processes Earth-observing satellite data and other sparse climate data inputs and generates complete models of the global climate that are the best fit of available data.

Examples of other workloads include the following:

- 3D modeling of high energy emission from rotation-powered pulsars
- 3D simulations of accretion to a star with magnetic field
- Assimilation of satellite observations of clouds to improve forecast skill
- Gravity wave simulations
- Global magnetohydrodynamic simulations of the solar wind in three dimensions

Like most other computational science and engineering workloads, NCCS Earth and space science applications represent the physical object of inquiry as a multidimensional grid and simulate the behavior of that object by computational manipulation of the grid. Climate models divide the Earth's atmosphere into cells and represent the behavior of wind, precipitation, clouds, heat, chemicals and other variables within and across cells by numeric simulation.

The NCCS' largest organizational system user is the Global Modeling and Assimilation Office (GMAO) [GMAO]. Currently, GMAO uses the "GEOS-5" code for its major production assimilation workload. GEOS-5 maps the Earth's surface using the Cubed Sphere, which is illustrated in Exhibit 1, below. The Cubed Sphere mapping avoids problems associated with more traditional mapping along lines of latitude and longitude (which suffer from very narrow cells near the poles that require special treatment). GEOS-5 allocates groups of adjacent atmospheric cells to hundreds or thousands of processors.

One can divide the GEOS-5 workload into physics and dynamics [GEOS-5]. Dynamics has to do with wind and air pressure. As wind and weather systems move across the Earth's surface, they cross the boundaries of simulated cells, so there needs to be communication across cell (and node) boundaries to reflect this. Physics includes the rest of the simulated variables and behavior, such as radiation, sea ice, atmospheric chemistry, moisture and vegetation. Physics is more local than Dynamics and makes better use of cache memory.

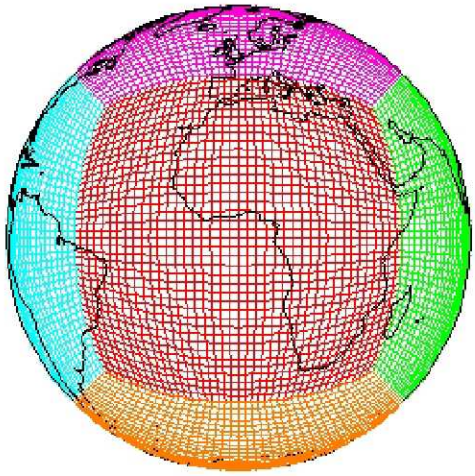


Exhibit 1 – Cubed Sphere Mapping of the Earth

1.3 CURRENT NCCS SYSTEMS

Currently, the principal computational platform at NCCS is "Discover," a Linux cluster that includes hardware manufactured by Linux Networx and IBM. Discover has 6784 CPUs, including dual- and quad-core Dempsey, Woodcrest, Harpertown and Dunnington processors manufactured by Intel[3]. Exhibit 2 summarizes the processing resources on Discover. Other Discover hardware resources include:

- Infiniband internal network mesh
- Disk drives from Data Direct Networks and other vendors
- Tape robots from Sun/StorageTek

[3] As of the paper submission date. Discover now includes Intel's Nehalem processors, as well.

Site	Goddard	Goddard	Goddard	Goddard
System	Discover - Base	Discover – Scalable Cluster Unit (SCU) 1&2	Discover - SCU 3&4	Dali – Data Analysis
CPU	Intel 5060 (Dempsey)	Intel 5150 (Woodcrest)	Intel 5420 (Harpertown)	Intel 7400 (Dunnington)
Clock - GHz	3.2	2.66	2.5	2.0
Release Date	May 06	June 06	Nov 07	Sep 08
MB L2 Cache/Core	2	2	3	1.5 MB L2 & 4 MB L3
Flops/Clock	2	2	4	4
Cores/Socket	Dual	Dual	Quad	Quad
Nodes/System	128	512	512	8 (4 Processors per Node)
Total Cores	512	2048	4096	128
Peak TeraFlops (TF) Calc	3.278	10.8954	40.96	1.02
GB Memory/Core	0.6	0.6	2	16
Front Side Bus MHz	1066	1066	1333	1066
Switch	Infiniband	Infiniband	Infiniband	Infiniband
OS	SUSE Linux	SUSE Linux	SUSE Linux	SUSE Linux
Scheduler	PBS	PBS	PBS	PBS
Message Passing Interface (MPI)	Scali-MPI	Scali-MPI	Open MPI 1.2.5	No MPI
Compiler	Intel Fortran 10.1.013	Intel Fortran 10.1.013	Intel Fortran 10.1.013	Intel Fortran 10.1.013
Manufacturer	LNXI	LNXI	IBM	IBM

Exhibit 2 Table of Discover Processor Components

This paper focuses on Discover's multi-core processor performance.

2. MULTI-CORE PROCESSOR DESIGN

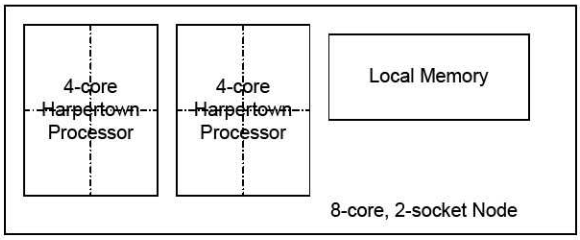
For the purposes of this paper, we distinguish between cores, processors and nodes.

- Cores = central processing units, including the logic needed to execute the instruction set, registers & local cache
- Processors = one or more cores on a single chip, in a single socket, including shared cache and network and memory access connections
- Node = a board with one or more processors and local memory, network attached

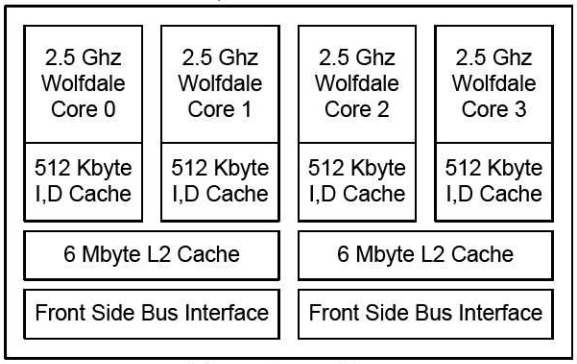
The upper conceptual (not physical) diagram in Exhibit 3 illustrates these terms.

Shown in the lower diagram are several levels of cache. Each Discover processor has two or three levels of cache memory. For example, the Harpertown quad-core processors have 512 Kbyte Level 1 (L1) instruction and data caches local to each of the 4 cores – each core has exclusive access to L1. There are also two 6 Mbyte Level 2 (L2) caches, each of which are shared by two cores – a total of 12 Mbytes of L2 cache on the processor. The Dunnington processors – the most recent ones added to the Discover complex – have a third level of cache as well, shared by all 4 cores on the processor. The next level of storage is main memory, 16 Gbytes per Harpertown node, and all cores and processors on the node share access to it. Note that each level of cache or main memory is larger and slower than the previous one.

The system user and job scheduler (on Discover, the Portable Batch Scheduler [Spear] from Altair) control



Cores, Processors and Nodes



Harpertown Processor

Exhibit 3 – Multi-Core Nodes and Processors

each application's use of one or more cores on the dual-core Dempsey and Woodcrest or quad-core Harpertown and Dunnington processors. When only one core on a processor and node are active, that core enjoys unencumbered use of all the processor and node resources, including all levels of cache, main memory and access paths. When more than one core is active, they must share these resources and contention occurs. We examine contention for the shared cache and main memory (collectively, the memory subsystem) in this paper.

3. MULTI-CORE BENCHMARK RESULTS

To begin with, we studied the performance of the memory subsystem with a single core active using kernel benchmarks.

The results that follow are variable in terms of memory *stride* and memory *range*. *Range* means the total memory footprint that the array variable called "cache" spans and the kernel touches. *Stride* means the distance between successive read and write [r+w] operations. The benchmark moves within the memory *range* and by the memory *stride* at each loop iteration.

"Cache miss latency" results are based on read operations; "Cache replacement time" results are based on write operations. R+w operations include one of each.

Note, some cluster systems, such as those manufactured by Silicon Graphics, provide direct access to all memory across the cluster from any core, processor or node. The Discover cluster, in contrast, and most current generation Linux clusters, only provide direct access to memory local to the node. Communication across nodes on Discover is handled by message passing using the MPI interface and the Infiniband internal mesh network. The simulation of Dynamics in GEOS-5 crosses node boundaries and uses the MPI interface. MPI also impacts multi-core performance, but is outside the scope of this paper.

3.1 MEMORY KERNELS

The memory core synthetic kernels are written in C and derived from codes published by [Hennessy] and [Mangegold]. The Hennessy code reads and writes data by incrementing an array variable, "cache," as seen in the following fragment. The Mangegold code functions similarly.

```
-----------------------------------------------------------------
/* inner loop does the actual read and write of memory
*/
for (j = 0; j < lim; j += stride) {
*cache[j]++; /* r+w one location in memory */
        } /* for j */
-----------------------------------------------------------------
```

3.1.1 SINGLE CORE KERNEL RESULTS

Exhibit 4 shows the kernel results using a single active core on the dual-core Woodcrest processor in the Scalable Cluster Unit 1&2 partition of Discover. (The stride length for Exhibits 4 through 7 is 256 bytes.)

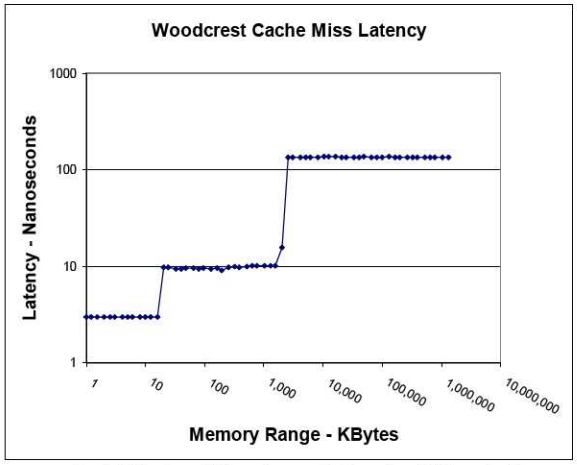


Exhibit 4 – Woodcrest Cache Miss Latency

Exhibit 4 can be interpreted as follows. The horizontal axis represents the range of memory crossed by the kernel program, from 1 Kbyte up to 10 Gbytes on a logarithmic scale. The vertical axis represents the cache miss latency (read time), also on a logarithmic scale, from 1 ns up to 1 µsec. The plot represents the latency time measured for read operations at a particular memory range.

Read operations covering a short range show the lowest latency, consistent with a high hit rate in the Level 1 cache. There is a distinctive stair-step pattern to the plot, consistent with latency times that jump up to a new plateau when the memory range exceeds the size of a particular level of cache. Cache replacement [write] time displays similar stair-step patterns. (Cache write results are omitted in this paper due to space limits.)

Observations about Exhibit 4 include the following. The points where latency jumps up correspond to the sizes of the L1 and L2 caches. The final plateau, starting at around 3 megabytes, corresponds to the latency of local main storage. The test does not extend beyond the capacity of local main storage and so does not reflect access times to remote storage (which would have to be accessed using MPI in any event).

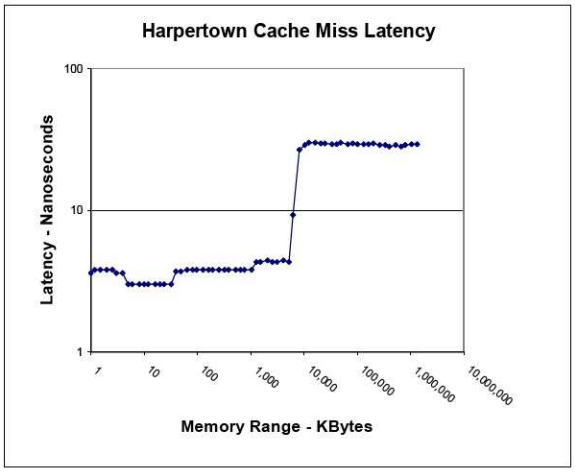


Exhibit 5 – Harpertown Cache Miss Latency

The format of Exhibit 5 is the same as for Exhibit 4, but shows the performance of the Harpertown chip. Comparing Woodcrest (a dual-core chip) to Harpertown (a quad-core chip with a slightly slower clock speed), the latency of Harpertown is somewhat faster than Woodcrest up to about the 2 Megabyte memory range, at which point the Woodcrest overflows its L2 cache and degrades to local main storage latency. When Harpertown overflows its L2 cache at around 6 Megabytes, it reaches a peak latency of around 30 ns. This performance is more due to the speed of the off chip main memory than to Harpertown itself. (The L1 cache stair step is modest for Harpertown with this test.)

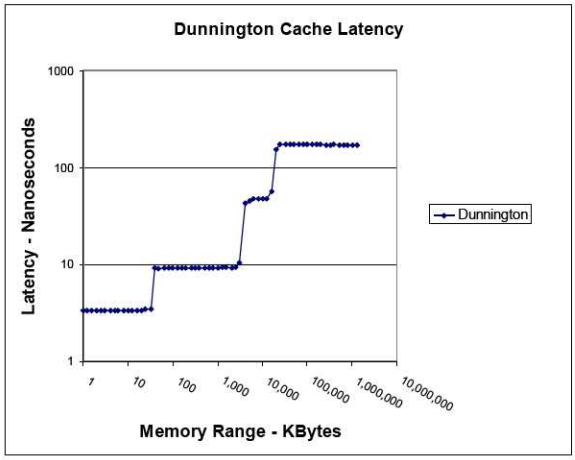


Exhibit 6 – Dunnington Cache Miss Latency

Exhibit 6 shows the same data in the same format for the Dunnington chip, which is of more recent vintage than either Woodcrest or Harpertown. Like the other results, it shows distinctive stair step performance as the kernel progressively overflows several levels of processor cache and eventually dips into local main storage. The extra stair step corresponds to the extra (L3) level of cache in Dunnington (see Exhibit 2).

Exhibit 7 is an overlay of the previous three charts. The larger and faster caches on Harpertown show that, except for small memory ranges (a few Kbytes), the newer Harpertown chip is faster than the older Woodcrest, notwithstanding the slower clock rate.

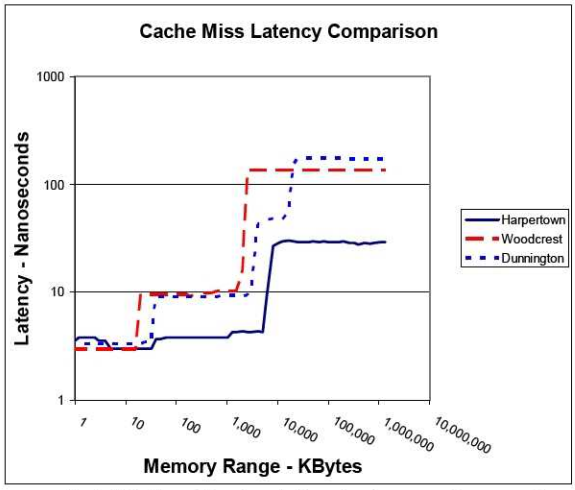


Exhibit 7 – Cache Miss Latency Comparison

Because of the transistor current leakage problem mentioned in the introduction, newer chips don't show the steady acceleration of clock speeds seen in prior generations. Improved performance and larger sizes of cache memory (and in the case of Dunnington, the newest processor in Discover, an additional level of cache) reflects the manufacturers' efforts to increase performance without increasing the clock rate.

The kernel benchmark is designed to eliminate the performance impact of each processor's execution units – which the clock rate would tend to emphasize. The code uses a dummy loop that executes instructions but does not stride through memory. The graphs show memory latency times calculated by subtracting the dummy loop execution time from the striding execution time. As a result, the kernel is narrowly focused on the performance of the on-chip caches and local main storage rather than the performance of the execution units.

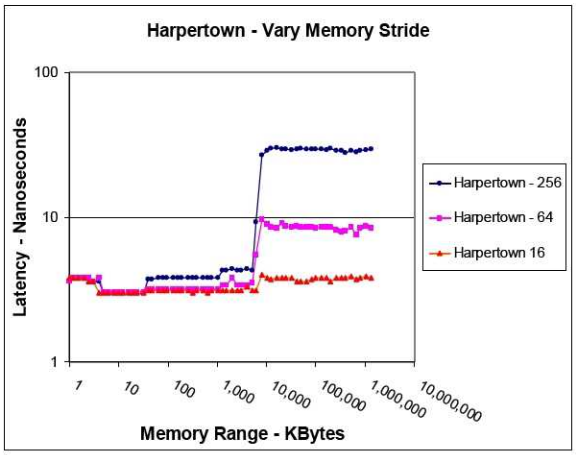


Exhibit 8 – Harpertown Cache Miss Latency With Varying Strides

Exhibit 8 shows additional results using the same memory kernel benchmark test, but varying the memory stride (the distance between consecutive read operations). For a given memory range, larger memory strides will cross that range faster than runs with smaller memory strides. The format of Exhibit 8 is the same as for the earlier charts, and all results are for the Harpertown chip, except that multiple memory stride distances are shown together.

Cache memories, as well as other storage hardware, improve computer performance due to the phenomenon of reference locality, which can be divided into temporal locality and spatial locality. Temporal locality occurs when a storage location used once is likely to be used again soon afterwards, so that holding the location's contents in fast cache storage will likely result in a cache hit. Spatial locality occurs when a storage location used once is likely to result in the use of a nearby storage location soon afterwards, so that holding a location and its near neighbors in fast cache storage will likely result in a cache hit.

In Exhibit 8, the best performance comes from the test run with a 16 byte memory stride. Due to spatial locality, the kernel experiences a very high hit rate when the code accesses near neighbor locations in rapid succession. For the 256 bytes stride, performance is significantly slower, with the characteristic stair-step pattern seen in the earlier graphics.

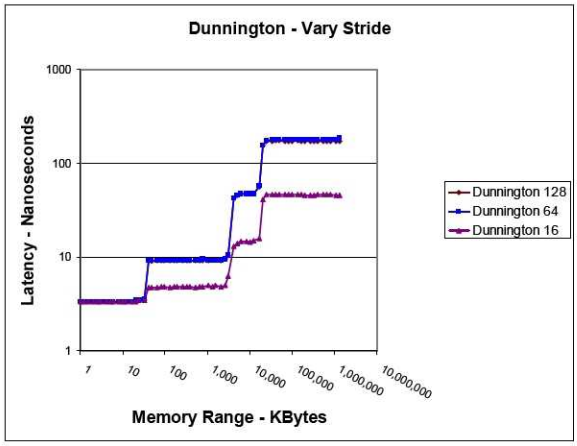


Exhibit 9 – Dunnington Cache Miss Latency With Varying Strides

Exhibit 9 shows results similar to Exhibit 8, but with the additional cache level stair step on the Dunnington processor. Note, in the exhibit, the Dunnington 128 and 64 byte stride results are nearly superimposed.

3.1.2 MULTI-CORE KERNEL RESULTS

Exhibit 10 is based on the Hennessy memory kernel code and is similar to the earlier exhibits, but slices the data in another way. One can read this Exhibit as follows. The horizontal axis is the memory stride size. As before, this axis has a logarithmic scale. The vertical axis is the read and write time, rather than just the read (latency) time, measured as before in nanoseconds; the range is from 0 to 700 nanoseconds and is not logarithmic.

More importantly, however, the graph shows the difference between 2-core, 4-core and 8-core performance. For each line, the core count is at the node level. The 2-core Woodcrest line means 2 cores

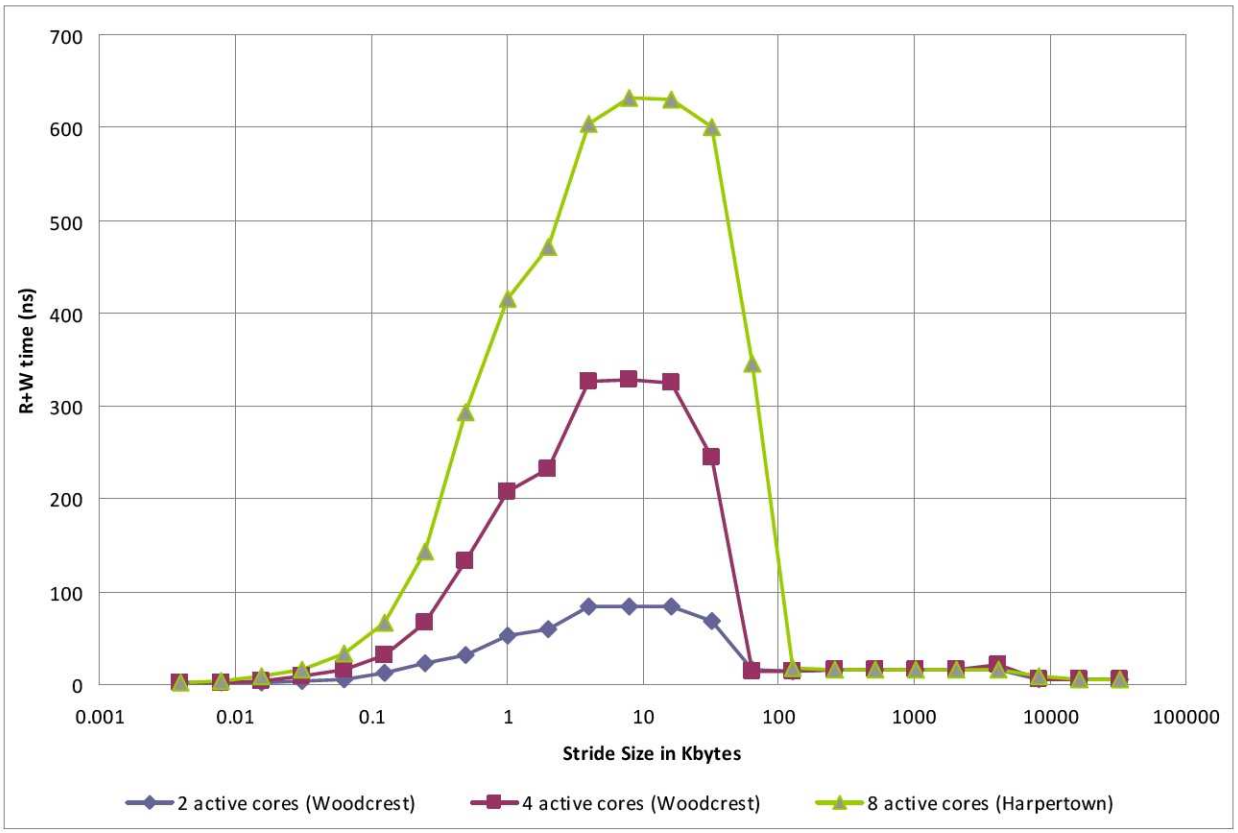


Exhibit 10 – Read and Write Time With Varying Strides and 2, 4 or 8 Active Cores

active on the node. Because there are 2 processors per node, this translates to 1 core active per chip. The 4-core line translates to 2 cores active per chip for the same reason, and the 8-core Harpertown line translates to 4 cores active per chip. (We used the results we had; all results from the same processor would have been more convincing but have the same basic pattern.)

Exhibit 10 demonstrates that these multi-core chips can experience contention for resources shared at the processor (chip) level. As the memory stride expands beyond sizes that ensure very high hit rates in the local cache, there is contention for shared cache, communications paths with the node and with main storage local to the node. So, going from single- to multi-core chips, in this case results in increased memory contention and reduced performance. The natural benchmark tests in the following section also show this.

Additional observations about Exhibit 10 include the following.

For small stride sizes, performance is very high for all core densities due to spatial locality. Performance degrades as the stride size expands and shows the same stair-step pattern as in the single-core test results that vary memory range.

On the right side of the graph, above 100 Kbyte stride sizes, performance improves dramatically for all core densities. The authors found that this improvement is most likely due to hardware prefetch. [Hegde] states, with respect to the Intel processor architecture, "The hardware prefetcher operates transparently, without programmer intervention, to fetch streams of data and instruction from memory into the unified second-level cache. The prefetcher is capable of handling multiple streams in either the forward or backward direction. It is triggered when successive cache misses occur in the last-level cache and a stride in the access pattern is detected, such as in the case of loop iterations that access array elements."

Aside from the programmer-transparent hardware prefetcher, the Intel instruction set has explicit user- and compiler-accessible prefetch instructions and vector instructions, both of which can avoid some of

the delays accessing the memory subsystem when there is spatial or temporal locality. Some compilers allow the application programmer to encourage the use of such instructions where the optimizer may not detect them automatically.

All of the above – multi-level cache, hardware prefetch, vector instructions and compiler optimizations – are examples of the exploitation of reference locality to improve system performance.

3.2 CUBED SPHERE BENCHMARK RESULTS

3.2.1 CORE DENSITY CUBED SPHERE RESULTS

This section shows results from running the Cubed Sphere application code on Harpertown processors within Discover, varying the number of cores, nodes and core densities.

Nodes	Cores Per Node	Total Cores	Wall Time (Seconds)
3	2	6	1336.9
3	4	12	771.2
6	2	12	676.1
3	6	18	658.8
3	8	24	601.3
6	4	24	411.6
12	2	24	371.1
6	6	36	339.2
6	8	48	318.3
12	4	48	212.8
12	6	72	181.9
12	8	96	178.5

Exhibit 11 – Cubed Sphere Results Varying Core Counts and Densities - Table

Exhibit 11 can be interpreted as follows.

Each Harpertown node on the Discover cluster system consists of two processors, plus local main storage and network connections with the rest of the system. Each processor chip has four cores, including L1 caches for data and instructions local to each core and a larger L2 cache shared by pairs of cores. With four cores per processor and two processors per node, there can be up to 8 cores active on a node during a program run. This core density and the total number of cores in the test is set by parameters passed to the job scheduler [Spear] when the job is submitted.

The first column in Exhibit 11 is the number of nodes in that run; the second column is the number of cores active per node; the third column is the total number of cores in the run and is the product of the first two columns. The fourth column is the wall-clock time.

Following are some observations about Exhibit 11.

As the total number of cores across all test runs increase, the wall clock time decreases – performance improves. Amdahl's Law states that the performance improvement possible running a program in parallel depends on the proportion of serial and parallel code in the program. In this case, the Cubed Sphere is very parallel, processing similar work for thousands of similar cells representing the Earth's atmosphere and oceans, so large performance improvements running on an increasing number of cores is not surprising. However, the observed speedup is less than linear - e.g., going from 12 to 24 processors at 2 cores per node improves performance from 676.1 down to 411.6 seconds – not nearly twice as fast. Cubed Sphere has *some* serial content.

On the other hand, as the core density per node increases, the wall clock time increases for a given total core count – performance degrades. For example, running 24 total cores on 12 nodes with a density of 2 cores per node results in a wall clock time of 371.1 seconds. Holding the total core count at 24 but increasing the core density per node to 4 (that is, 2 cores active on each processor) and reducing the number of nodes to 6 gives a wall clock time of 411.6 seconds. Increasing core density to 8 cores per node (all that are available) increases the wall clock time to 601.3 seconds.

All of these 24-core test runs had the same problem size and the same execution unit resources available. The difference was in the shared resources. Going from 2 to 4 core density per node meant increased contention at the chip level – contention for the chip's access paths to main storage and for the use of main storage, resulting in approximately a 10% increase in wall-clock time. Going from 4 to 8 core density means contention for the shared Level 2 cache as well as main storage, and results in a 50% wall clock time increase.

So far, our benchmark results have shown (a) that increasing core counts increases performance, (b) that this performance increase is less than linear and (c) that increasing core density for a fixed core count decreases performance. The results that follow lead to some additional conclusions, but as they do not

vary the core density, they neither confirm nor deny conclusion (c).

3.2.2 HIGH CORE COUNT CUBED SPHERE RESULTS

The following charts show Cubed Sphere results on large numbers of processor cores [Putman]. In these charts, core density is held at the maximum available on the node – e.g., 8 cores per node for Harpertown processor nodes on Discover.

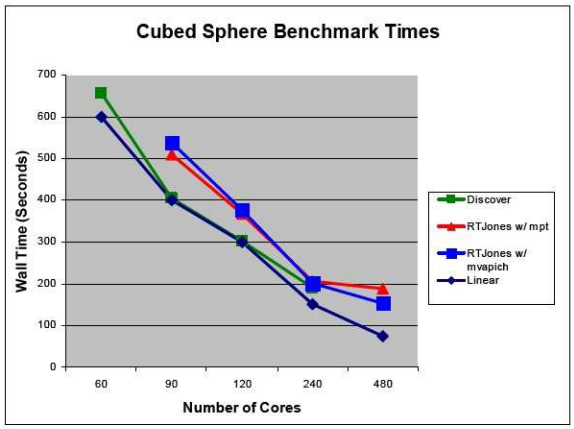


Exhibit 12 – Cubed Sphere – Sub- and Super-Linear Speed-Up

The format of Exhibit 12 is as follows. The horizontal axis is the number of cores running the test. The vertical axis is the wall clock time in seconds. The line labeled "Linear" represents the theoretical linear speed-up achievable from a 60-core run extrapolated to a 480 core run.

The "Discover" line is for the NCCS' Discover cluster running on Harpertown processors. The two "RTJones" lines are for the RTJones cluster supercomputer located at the NASA Ames Research Center in California. There are two RTJones lines, as the test was run using two different versions of the MPI message passing software.

Observations about Exhibit 12: Discover benchmark results showed a slightly better than linear speed up in the range between 60 and 90 cores. Between 90 and 120 cores, there was a close match between the Discover and theoretical linear speedup; and above 120 cores, the speed up was not a good as the theoretical linear improvement.

The behavior of the RTJones tests with respect to speedup was similar to that for Discover in Exhibit 12 – varying from slightly better to slightly worse than theoretical

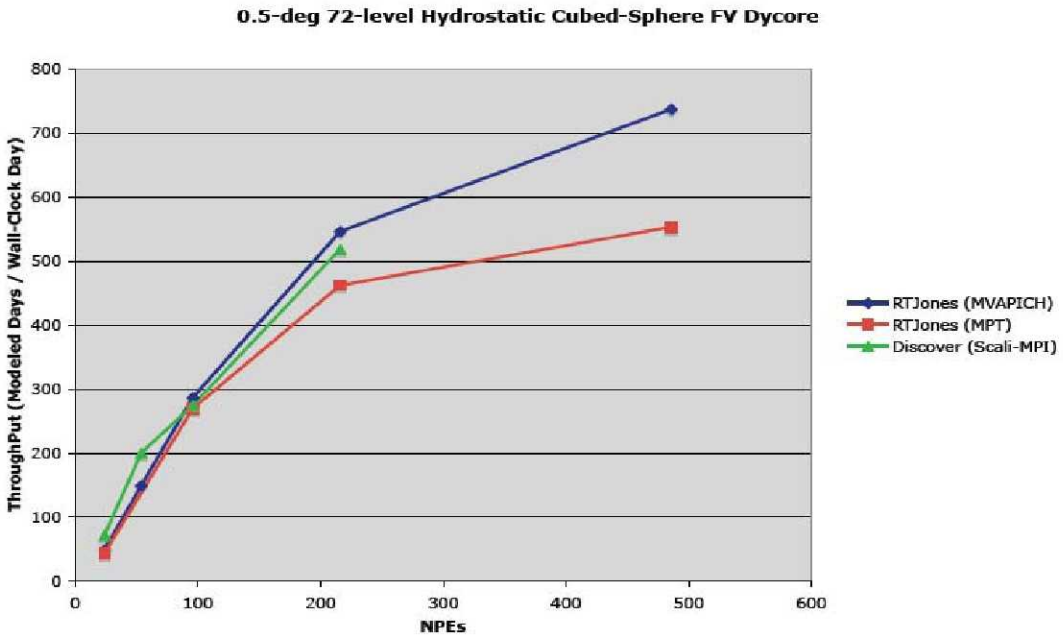


Exhibit 13 – Throughput

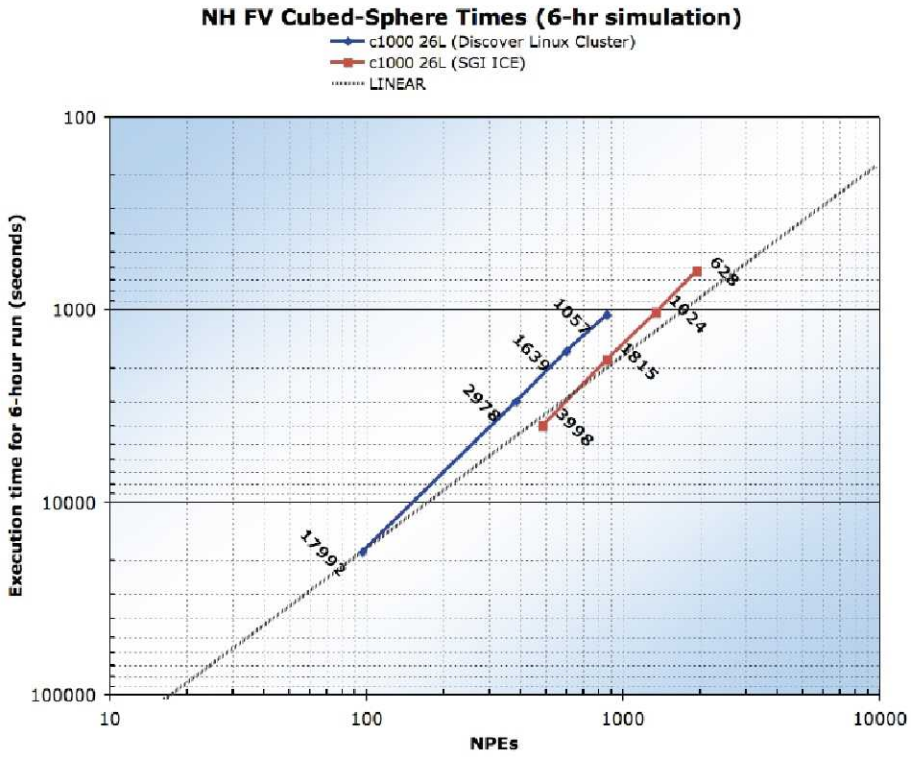


Exhibit 14 – Super-Linear Speedup

performance changes as the number of cores increased. In addition, running two different version of MPI showed that there was some performance impact from changing the MPI version. Overall, Discover was faster than RTJones.

Exhibit 13 also compares Discover with RTJones. The horizontal axis is the same – the number of cores (here labeled "NPEs" – the number of processing elements) working on the problem. However, the vertical axis is inverted from the prior graph. In Exhibit 13, the vertical axis is Throughput rather than Wall Clock time – higher up is faster and better.

For all three test runs in Exhibit 13, one can see that throughput improvements tail off as the number of cores (NPEs) increases, most noticeably above 200 cores. Up to 100 cores, RTJones is faster; above that, Discover is faster. As in Exhibit 12, there is a slight but noticeable difference between the two MPI versions running on RTJones. In both Exhibits 12 and 13, the performance line becomes more horizontal as the core count increases – indicating less additional performance for more processors.

Note, Exhibits 12 and 13 both show Cubed Sphere benchmark results, but in Exhibit 13 it is a subset of the code rather than the whole program. Exhibit 13 shows results running the GEOS-5 Dynamics but not Physics.

Exhibit 14 also shows Cubed Sphere benchmark results, but has a different format. The horizontal axis is logarithmic and shows the number of processor elements (cores). The vertical axis is execution time, uses a logarithmic scale and displays the longest time lowest on the scale. The Exhibit shows more clearly than the others that for a fixed core density and large core counts, Cubed Sphere performance improves better than a theoretical linear extrapolation. A paper by Shameem Akhter [Akhter] excerpted below provides an explanation.

4. OTHER MULTI-CORE TEST RESULTS

There is a substantial literature and some controversy surrounding multi-core processor performance [Akhter], [Alam], [Chai], [Dongarra], [Levesque], [Mangegold], [Simon].

Aside from the science and engineering supercomputer niche, multi-core processors have penetrated many other markets, including mainframes, servers, desktops and laptops. Multi-

core is the predominant processor architecture in these markets today. Manufacturers of all these types of systems have embraced multi-core and these diverse platforms have experienced the multi-core memory contention problem.

A sample from this literature is given in the Bibliography and in the following discussion.

LEVESQUE

In this paper [Levesque], based on analysis of Opteron processors on Cray cluster systems, the authors state that "excluding message passing performance, the primary source of contention when moving from single core to dual core is memory bandwidth." Using the NERSC-5 SSP applications, Levesque found an average performance penalty moving from single to dual core of 10%; using the NAS Parallel benchmarks, the penalty ranged between 10% and 45%.

Levesque proposes a simple model for predicting multi-core application performance. The model divides total execution time into time spent on shared resources (memory bandwidth) and non-shared resources (everything else). The model predicts that memory bandwidth time will double going from single to dual core, and double again going from dual to quad core. Levesque's own tests show that the model's predictions are quite accurate. Our results, e.g. for the 24 core results shown in Exhibit 11, are consistent with this model as well.

AKHTER

This paper [Akhter] discusses Gustafson's Law, a variation on Amdahl's Law that eliminates certain assumptions, such as the assumption that a problem size is fixed. "Amdahl's Law assumes that as the number of processor cores increases, the problem size stays the same. In most cases, this is not valid. Generally speaking, when given more computing resources, the problem generally grows to meet the resources available." As noted in an earlier CMG paper [Glassbrook], climate models running on NCCS systems follow this pattern, exploiting increasingly parallel machines to run at finer resolutions (smaller cell sizes) and achieving better scientific results thereby.

Ahkter's discussion of Gustafson's Law touches an observed characteristic of the Cubed Sphere test, where high core counts in a certain range show better than linear performance improvements due to improved cache utilization. "Amdahl's Law assumes that the best performing serial algorithm is strictly limited by the availability of CPU cycles. This may not be the case. A multi-core processor may implement a separate cache on each core. Thus, more of the problem's data set may be stored in cache, reducing memory latency." Putman's test results (Exhibit 14) on Discover and RTJones demonstrate this phenomenon.

5. CONCLUSIONS AND COMMENTS

These NCCS benchmark results, results published elsewhere, analysis of the NCCS' workload and other workloads and of the Discover hardware design led to these conclusions:

- Multi-core processors work both in favor of and against improved application software performance. Positive and negative performance impacts included the following.

- Contention for shared resources by multiple cores on the same processor (which may include the Level 2 cache, the path to node- and cluster-level resources, local main storage and off-node storage) will reduce wall-clock performance compared to a single core on the same processor (Exhibit 11).

- Spreading a fixed-size workload across multiple cores may decrease the working-set size for each workload segment, increase spatial reference locality, increase cache hit rates, decrease the need for communication with other cores, processors or nodes, decrease contention for shared memory access and so improve wall clock performance (Exhibit 14 & Exhibits 12 and 13 below 200 cores).

- Increasing the number of cores allocated to a fixed-size workload that can run in parallel will decrease the amount of local computation needed and so improve performance (Exhibit 14 & Exhibits 12 and 13 below 200 cores.

- For the Cubed Sphere, the improvements due to smaller working sets and decreased local computation can be overcome by other factors. The authors believe that the tail-offs seen in Exhibits 12 and 13 above 200 cores are due to increase contention for the MPI message passing resource across nodes. Eventually, the increasing cost of MPI communications across increasing core counts (e.g., as simulated weather crosses node boundaries) overcomes the

decreasing cost of local computation and higher cache hit rates.

- The GEOS-5 Cubed Sphere climate data assimilation code is highly susceptible to parallel execution, as the treatment of different areas of the Earth's surface, oceans and atmosphere is quite similar. GEOS-5 is an excellent example of an application that can make good use of a highly parallel Linux cluster HPC system.

- It's not sufficient to simply run an application on more cores to achieve better performance. Users should examine their codes and consider restructuring them to increase locality, increase intra-node communications, use MPI functionality to promote spatial locality, use compiler optimizations and make other multi-core aware changes. Insight into multi-core specifically and processor and cluster design generally can help application performance. Microprocessor designs such as Intel's and AMD's have different application performance implications.

- Where run times are the critical constraint and processor utilization is less critical, running single-core density can help performance. The scheduler can support single-core runs.

- Quad-core+ multi-socket nodes will likely exacerbate the bandwidth contention issue both to main memory and on-chip over the middle term. Many applications will experience significant negative multi-core processor performance impacts unless optimized to account for them. Processor designers are responding to this problem.

BIBLIOGRAPHY

[Akhter] Akhter, Shameem and Roberts, Jason. "Multi-Core Programming – Increasing Performance Through Software Multi-threading." Intel Press, April 2006.

[Alam] Alam, S.R., Barrett, R.F., Kuehn, J.A., Roth, P.C., and Vetter, J.S. "Characterization of Scientific Workloads on Systems with Multi-Core Processors." IEEE International Symposium on Workload Characterization. Oct. 2006, San Jose, CA; 225-236.

[Chai] Chai, L., Gao, Q., and Panda, D.K. "Understanding the Impact of Multi-Core Architecture in Cluster Computing: A Case Study with Intel Dual-Core System." International Symposium on Cluster Computing and the Grid, 2007, Rio de Janeiro, Brazil, 2007.

[Dongarra] Dongarra, J., Gannon, D., Fox, G, and Kenned, K. "The Impact of Multicore on Computational Science Software," CTWatch Quarterly, 2007.

[Glassbrook] Glassbrook, Richard and McGalliard, James. "Performance Management at an Earth Science Supercomputer Center." CMG 2003.

[GEOS-5] gmao.gsfc.nasa.gov/systems/geos5

[GMAO] gmao.gsfc.nasa.gov

[Hennessy] Hennessy, J. and Patterson, D. Computer Architecture: A Quantitative Approach, 2nd Edition. Morgan Kauffmann, San Mateo, California.

[Hegde] Hegde, Ravi. "Optimizing Application Performance on Intel® Core™ Microarchitecture Using Hardware-Implemented Prefetchers," software.intel.com.

[Levesque] Levesque, J., Larkin, J., Foster, M., Glenski, J., Geissler, G., Whalen, S., Waldecker, B., Carter, J., Skinner, D., He, H., Wasserman, H., Shalf, J., Shan, H., and Strohmaier, E. "Understanding and Mitigating Multicore Performance Issues on the AMD Opteron Architecture" (March 7, 2007) Lawrence Berkeley National Laboratory. Paper LBNL-62500.

[Mangegold] Mangegold, Stefan; Boncz, Peter; and Kersten, Martin. "Optimizing Main Memory Join on Modern Hardware," Technical Report INS-R9912, CWI (Centre for Mathematics and Computer Science). Amsterdam, 1999.

[Moore] http://en.wikipedia.org/wiki/Moore%27s_law

[Putman] Putman, William M. "The Finite-Volume Dynamical Core on the Cubed-Sphere" Poster. NASA-NOAA-Florida State University, 2007, and email correspondence.

[Simon] Simon, Tyler; Cable, Sam; and Mahmoodi, Mahin. "Application Scalability and Performance on Multicore Architectures," HPCMP Users Group Conference, IEEE Computer Society, 2007.

[Spear] Spear, Carrie and McGalliard, James. "A Queue Simulation Tool for a High Performance Scientific Computing Center." CMG 2007.